ARMY RESEARCH LABORATORY ARL

Implementation of a Cascaded Histogram of Oriented Gradient (HOG)-Based Pedestrian Detector

by Christopher Reale, Prudhvi Gurram, Shuowen Hu, and Alex Chan

ARL-TR-6661 September 2013

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-6661 **September 2013**

Implementation of a Cascaded Histogram of Oriented Gradient (HOG)-Based Pedestrian Detector

Christopher Reale, Prudhvi Gurram, Shuowen Hu, and Alex Chan
Sensors and Electron Devices Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2013	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** 05/2012 to 05/2013
4. TITLE AND SUBTITLE Implementation of a Cascaded Histogram of Oriented Gradient **(**HOG)-Based Pedestrian Detector		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Christopher Reale, Prudhvi Gurram, Shuowen Hu, and Alex Chan		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-SES-E 2800 Powder Mill Road Adelphi, MD 20783-1197		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-6661
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this report, we present our implementation of a cascaded Histogram of Oriented Gradient (HOG) based pedestrian detector. Most human detection algorithms can be implemented as a cascade of classifiers to decrease computation time while maintaining approximately the same performance. Although cascaded versions of Dalal and Triggs's HOG detector already exist, we aim to provide a more detailed explanation of an implementation than is currently available. We also use Asymmetric Boosting instead of Adaboost to train the cascade stages. We show that this approach reduces the number of weak classifiers needed per stage. We present the results of our detector on the INRIA pedestrian detection dataset and compare them to the results provided by Dalal and Triggs.

15. SUBJECT TERMS
Human detection, pedestrian detection, cascade detector

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Shuowen Hu
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	36	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-2526

Contents

List of Figures

1. Introduction

The general problem of detecting humans in images and videos has been heavily researched over the past decade. There are many applications that require accurate and efficient human detectors with the most obvious being automatic visual surveillance, in which the presence and behavior of individuals in an area of interest is closely monitored. Given the steep increase in asymmetric and urban warfare in the past decade, efficient and accurate dismount detection has become an essential operational capability in the military arena. Another important application of human detector is in the field of robotics, as many robotic technologies require smooth interactions between robots and humans. One way for robots to recognize humans is to detect them with visual sensors (cameras). Yet another application is a computer-driven car. Safety issues represent a significant portion of the concerns when designing an autonomous car. Detecting, locating, and avoiding nearby pedestrians is paramount to addressing these concerns. While on the surface this problem may seem easily solvable, the large variability in appearance due to clothing, physique, pose, and viewpoint make human detection challenging. For example, a young girl wearing a swimming suit has a vastly different appearance than a bearded man dressed in hunting gear. Both of them will look radically different from a football player viewed from his side. Recognizing humans across this vast spectrum of appearances is a very difficult task.

While researchers have taken several approaches to detecting humans, the most common is the sliding window technique. The sliding window approach involves developing a detector for a fixed-size window and then evaluating that detector on a dense grid of locations throughout the scale and space of the image. Due to the large number of evaluation locations for a given image, a detector must be very fast to operate on video in real time. Even a small improvement in speed for a single detection window can result in significantly less computation time for the evaluation of an entire image or video.

In this report, we present an implementation of a sliding window pedestrian detector. Our detector is based on the work of Zhu et al. (*1*). Our method most notably differs from their method in the boosting algorithm employed. Zhu et al. use AdaBoost, while we use the more general Asymmetric Boosting, which optimizes for arbitrary Type 1 and Type 2 error penalties (*2*). This difference enables us to achieve similar detection and false positive rates in a cascade stage while using fewer weak classifiers, thus decreasing the run time on test images. In actuality, we were not able to replicate the results presented by Zhu et al. and our corresponding AdaBoost detector performed worse than their reported results. While we could not achieve their reported performance, we were able to improve upon our implementation of Zhu's algorithm. Our improvement could potentially transfer to a better original implementation. The rest of the report is organized as follows: section 2 contains descriptions of works related to ours, section 3 contains implementation details, section 4 contains experiments and results, and section 5 concludes the report.

2. Related Work

Pedestrian detection has been studied for the better part of the past 20 years and we briefly describe the works most closely related to our detector. Perhaps the most well-known human detection algorithm was presented by Dalal and Triggs (*3*). Their method involves training a linear Support Vector Machine (SVM) on a dense grid of Histogram of Oriented Gradient (HOG) features. The main contribution of their method is the use of HOG features in human detection. They studied the effects of altering ways to calculate and normalize the features and presented their results. Zhu et al. increased the speed of the Dalal and Triggs method while maintaining a comparable performance. They train a cascade with AdaBoost to prune out negative samples. Their cascade learning algorithm was inspired by the Viola and Jones object detector (*4*). Viola and Jones used a set of simple Haar-like features to perform face detection.

As with human detection, there has been significant investigation into boosting. Boosting was first introduced by Schapire and Freund (*5*) and has been a powerful tool frequently used in the machine learning community ever since due to its simplicity and performance. Boosting works by iteratively training a weak classifier using a machine learning algorithm, adding it to the set of previously trained weak classifiers, and then updating the distribution of the training data based on the output of the newest weak classifier. After each iteration, samples that were misclassified in previous iteration are given more weight and samples that were classified correctly are given less weight. That way, samples that are harder to classify will receive more “attention” from the training algorithm. A statistical explanation for the excellent performance of AdaBoost and its tendency of not overfitting was given by Friedman et al. (*6*), which recasts boosting as a regression problem.

In a cascade detector, the objective is to train classifiers that have asymmetric error rates (i.e., different false detection and miss rates). There have been many attempts to generalize AdaBoost to solve this problem. We use a method proposed by Masnadi (*2*), which extends the statistical formulation of Friedman.

3. Implementation

3.1 Operation

Like many detection algorithms, our human detector adopts the sliding window approach. That is, a single detector is trained using fixed-size image patches (128 x 64 pixels) extracted from a dense set of overlapping locations (windows) throughout the input image at different scales. This is realized by first down-sampling the image several times to create an image pyramid and then training and evaluating the detector at a dense grid of locations within each layer of the pyramid. We downsize the image by 5% between each level of the pyramid, yielding approximately 14 levels per octave.

We evaluate each window using a cascaded detector. As shown in figure 1, a cascaded detector is a sequence of strong classifiers (stages) that attempts to reject negative samples while allowing positive samples to pass through. For a sample to be detected as a human, it must be deemed a human by every stage of the cascade. To account for this structure, the cascade stages are designed in such a way that there is a very low probability of a stage missing a detection, while there is a fairly high probability of identifying a negative sample as a human. This structure allows the cascade to operate more efficiently as well. The vast majority of samples evaluated by a detector will be negative, therefore filtering them out quickly greatly improves the overall speed of the detector. While it may perform slower on positive samples, there are so few of them in an image that the running time of the cascade will not be noticeably affected. We experimented with cascade architectures that have 20 to 40 stages in complexity.

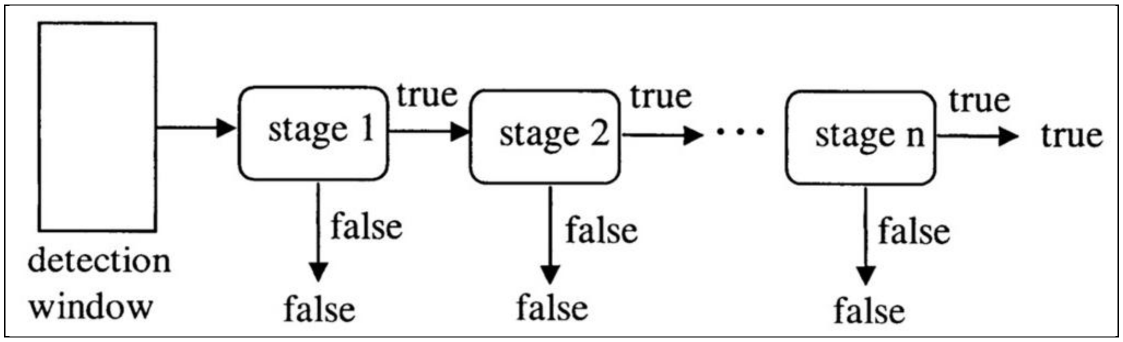

Figure 1. Structure of a cascade detector.

Each cascade stage (strong classifier) is itself comprised of a set of SVMs (weak classifiers). Each weak classifier operates on the HOG features of a given sub-window (block) within the window. The weak classifiers are combined by weighting and adding their binary outputs and then thresholding them to generate a binary decision. This (arithmetic) method of combining the weak classifiers differs from the logical method of combining the strong classifiers.

One last step that must be performed is non-maximal suppression. Typically, the detector yields several overlapping detections near the location of a single pedestrian. This naturally occurs due to the dense sampling of evaluation windows throughout the images. Obviously, it is more desirable to select only the best detection for each target. We use the non-maximal suppression algorithm of Dollar et al. (*7*) to perform this task and obtain a final detection.

3.2 Feature Extraction

We use HOG features as described by Dalal and Triggs, which are closely related to the Scale Invariant Feature Transform (SIFT) described by Lowe (*8*). Our HOG features are all extracted from blocks containing four cells configured in a 2 by 2 layout. The blocks can have one of three aspect ratios: 2x1, 1x1, or 1x2. The cells take on the aspect ratio of the block to which they belong. We use a very large set of blocks of all sizes to train the cascade. While not every feature is used, the training algorithm has 5028 blocks available to it, ranging from 12 by 12 to 128 by 64 in size.

Features are extracted by first calculating the gradient at each pixel and a set of HOG features is calculated for each cell. A gradient is binned according to its orientation (we use 9 bins over 180°) and the weight it contributes is proportional to its magnitude. As described by Dalal and Triggs, normalization of the histograms within a block is crucial to the effectiveness of the features. Therefore, we normalize the features for each block using the 1-norm. Thus, for each block, we calculate a 36-dimensional feature with unit 1-norm.

3.3 Integral HOG Image

In order to enable the quick calculation of features, we implemented a data structure called an Integral HOG Image. As described by Zhu et al., based on the work done by Viola and Jones, the Integral HOG Image enables the HOG feature for any rectangular cell to be calculated very quickly. At each pixel, the Integral HOG Image stores the raw (not normalized) HOG feature of the region above and to the left of the corresponding pixel in the original image. As shown in figure 2, once the Integral HOG Image is constructed, a HOG feature for a given region can be calculated by adding four histograms and then normalizing.

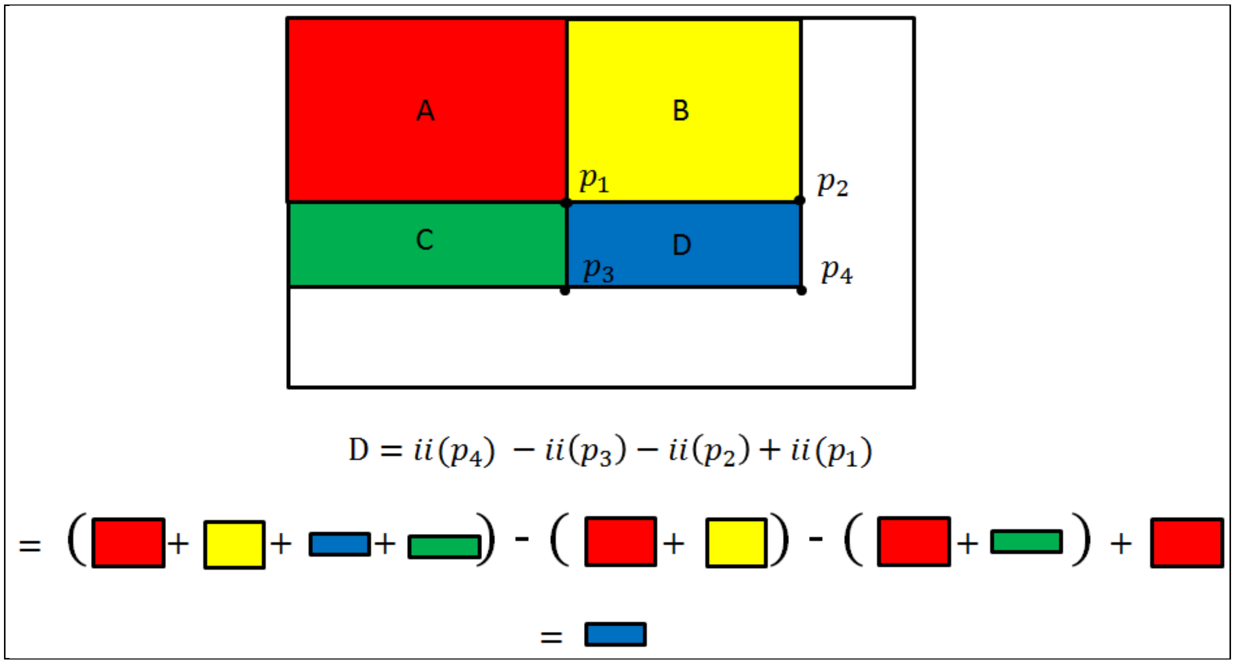

Figure 2. Efficient computation of HOG features from an integral HOG image.

3.4 Training

As stated previously, the objective of the cascade structure is to reject as many negative samples as possible, while allowing almost all positive samples to pass through each stage. This translates to each stage having an extremely high detection rate with a moderate false positive rate. Each stage achieves this objective by using Asymmetric Boosting.

We begin the training of each stage by obtaining sets of positive and negative samples. We initialize the distribution of the samples to be uniform. In each iteration of the boosting algorithm, we randomly select a set of 125 blocks and train a linear SVM on the features extracted from their corresponding locations in the training samples. We slightly alter the standard SVM algorithm to handle unequal weights in the sample distribution. We add the best feature to the current classifier, update the weights of the samples, and continue to the next iteration. This algorithm runs until the combined weak classifiers achieve a certain performance metric (high detection rate, moderate false positive rate) on the training set. When this occurs, the strong classifier is considered to be trained. Figure 3 depicts a flowchart of the training algorithm. The pseudocode for the boosting portion of the algorithm is also shown in figure 3.

```
**Algorithm: Asymmetric Boosting**
  Given training data $(x_1, y_1) \cdots x_n, y_n)$ where $y \in \{1, -1\}$
  Initialize training sample weights to uniform. $w_i = \frac{1}{2|I_+|} \forall i \in I_+, w_i = \frac{1}{2|I_-|} \forall i \in I_-$
  Choose penalties for different error types $C_1$ and $C_2$
  Do
    Train 125 SVMs using the current sample weights
    For each SVM $h_j$
      Calculate $T_+ = \sum_{i \in I_+} w_i$, $T_- = \sum_{i \in I_-} w_i$, $b = \sum_{i \in I_+} w_i 1(y_i \neq h(x_i))$, and $d = \sum_{i \in I_-} w_i 1(y_i \neq h(x_i))$
      Calculate $a$ by solving $2C_1 b \cosh(C_1 a) + 2C_2 d \cosh(C_2 a) = C_1 T_+ e^{-C_1 a} + C_2 T_- e^{-C_2 a}$
      Calculate loss $l = (e^{C_1 a} - e^{-C_1 a}) b + T_+ e^{-C_1 a} + (e^{C_2 a} - e^{-C_2 a}) d + T_- e^{-C_2 a}$
    Add SVM/step-size $(h_m(x), a_m)$ with smallest loss to strong classifier
    Update weights by $w_i = \begin{cases} w_i e^{-C_1 a_m h_m(x_i)}, & i \in I_+ \\ w_i e^{C_2 a_m h_m(x_i)} & i \in I_- \end{cases}$
  While(Strong classifier doesn't meet desired performance)
  Final classifier: $f(x) = \mathsf{sign}(\sum_{m=1}^{T} a_m h_m(x))$ where $T$ is the number of weak classifers
```

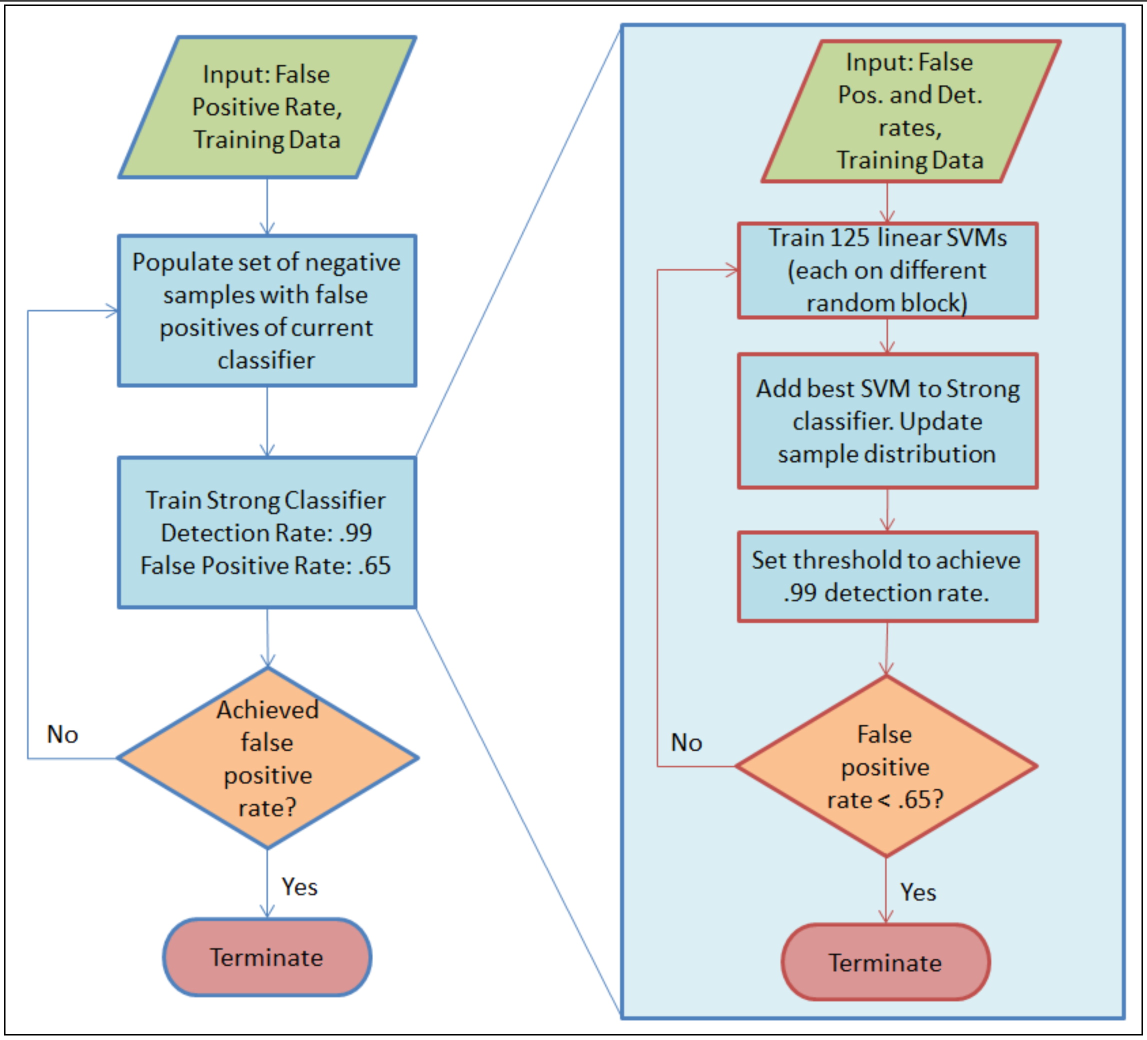
Input: False Positive Rate, Training Data
Populate set of negative samples with false positives of current classifier
Train Strong Classifier Detection Rate: .99 False Positive Rate: .65
Achieved false positive rate?
No
Yes
Terminate
Input: False Pos. and Det. rates, Training Data
Train 125 linear SVMs (each on different random block)
Add best SVM to Strong classifier. Update sample distribution
Set threshold to achieve .99 detection rate.
False positive rate < .65?
No
Yes
Terminate

Figure 3. High-level flowchart of the cascade training algorithm.

As in Dalal and Triggs and Zhu et al., we train on the INRIA dataset. We use all of the positive samples (and their horizontal mirrors) to train each stage. We choose negative samples that have not been filtered out by any of the previously trained stages. This is analogous to the mining of "hard negatives" in Dalal and Triggs to improve performance.

One advantage of this training process is that it is incremental. That is, if a 20-stage classifier is trained and a 30-stage classifier is needed, it can be built off of the 20-stage classifier rather than starting from scratch. The incremental aspect is also useful for evaluation. Suppose a 40-stage classifier is trained but the detection rate is not high enough. Stages of the classifiers can simply be removed so as to improve the detection rate (at the expense of a larger false positive rate). This is an effective way to generate a receiver operating characteristic (ROC) curve for the detector (as opposed to the standard varying threshold method).

Our implementation of the training algorithm can be founded in the appendix.

4. Experiments

4.1 Classifier Reduction

The main difference between our method and Zhu's method is the use of Asymmetric Boosting to reduce the number of classifiers. We compare our Asymmetric Boosting based algorithm against a cascade trained using AdaBoost. As shown in figure 4, Asymmetric Boosting trains cascade stages with a smaller number of classifiers but attains comparable performance. The reduction in weak classifiers directly corresponds to a reduction in computation time of the cascade. Thus, the cascade can process more images or video at a faster frame rate.

While we were able to decrease the number of weak classifiers used in the strong classifier training, we were unable to match the results published by Zhu. Therefore our classifier does not match the speed of Zhu's classifier (though it does outperform *our implementation* of Zhu's classifier). We are currently investigating the causes of this discrepancy.

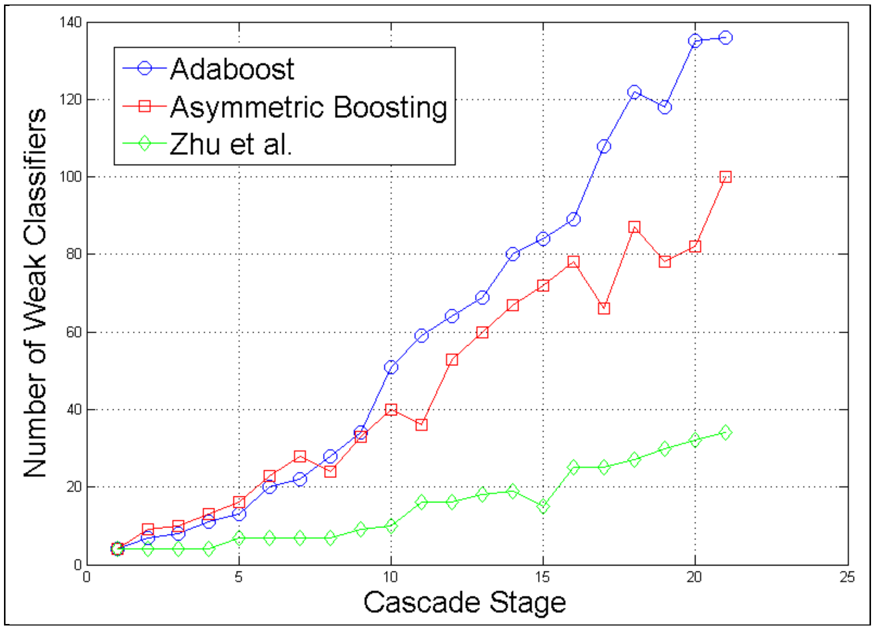

Figure 4. Comparison of the number of classifiers used in each stage when using AdaBoost and Asymmetric Boosting.

4.2 Performance

We encountered problems with replicating the accuracy of Zhu's classifier as well. Their classifier achieves comparable performance to the Dalal and Triggs detector. We evaluated our algorithm and the Dalal and Triggs algorithm on the testing portion of the INRIA pedestrian detection dataset. As shown in figure 5, the results of our classifier with various parameter settings do not match up to HOG. At 0.1 false positives per image (FPPI) rate, the Dalal and Triggs detector has a miss rate of 0.49 while our detector has a miss rate of 0.67. We are currently investigating the cause of this discrepancy.

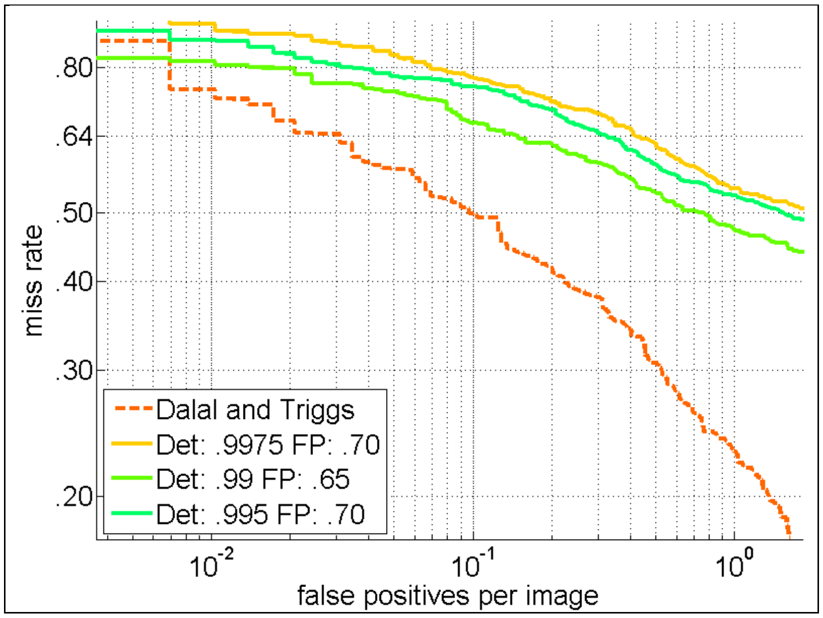

Figure 5. Performance of our algorithm and the Dalal and Triggs algorithm on the INRIA pedestrian detection dataset.

4.3 Sample Results

In order to give a better feel for how our detector performs, we provide a few example images with detection results. Note that these results have not been processed by non-maximal suppression so as to illustrate the behavior of the cascade.

The first image, shown in figure 6, shows the classifier producing the desired results. There are four people in the image and the classifier manages to detect each one of them. The classifier does not detect humans at any other locations, thus there are no false positives. In general, the detector does well in images without much clutter in the background.

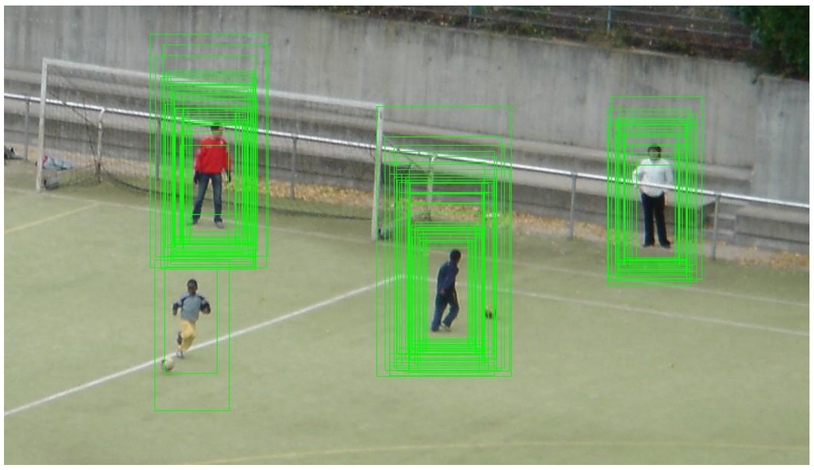

Figure 6. Example of successfully human detections without any false positives.

The second image, shown in figure 7, shows the classifier producing fairly good, but not ideal results. There are two people in the image. Depending on the configuration of the non-maximal suppression algorithm, the person on the left may or may not be detected. This also illustrates the importance of choosing a good non-maximal suppression algorithm. Note that some methods of non-maximal suppression may work better for some detection algorithms and worse for others.

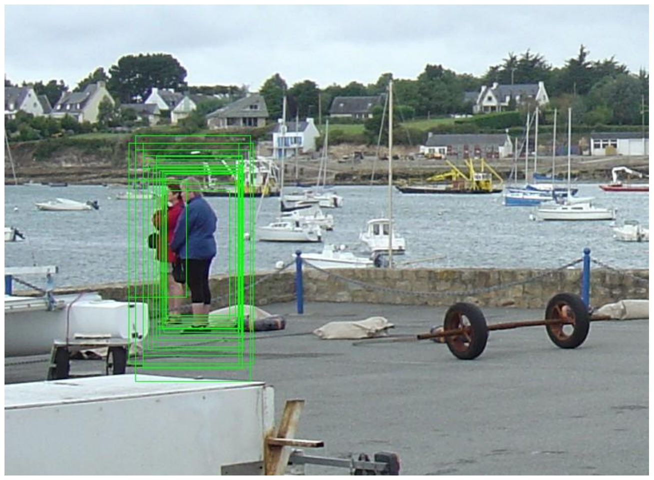

Figure 7. Example of human detections without strong false positives.

The third image, shown in figure 8, shows the classifier producing poor results. In this image, there are two people. While the detector manages to detect both of them, it also produces several false positives. This tends to be a common theme in many test images (not shown); if false positives exist in an image, there tend to be many of them. Urban scenes tend to exhibit this problem more than natural scenes. A large number of false positives in a small percentage of images could be one reason why our results are poor. Figure 8 also shows how our detector has mistakenly detected cars, mainly due to some competing car structures that incur similar HOG features as those produced by signatures of pedestrians.

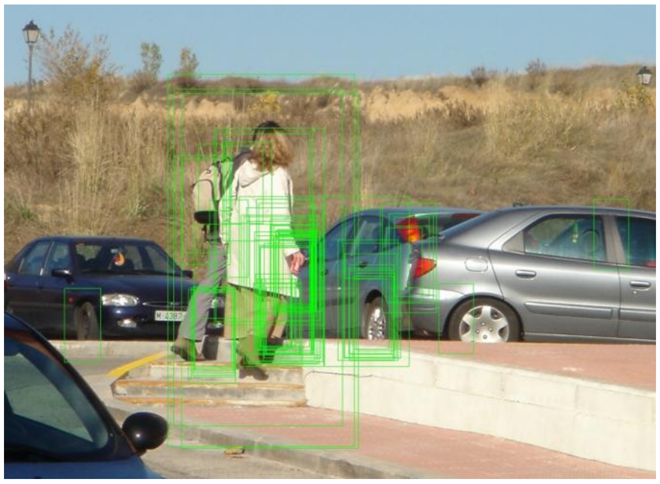

Figure 8. Example of human detections with several false alarms.

5. Conclusion

In conclusion, we have described in detail our implementation of a cascaded HOG-based pedestrian detector, from feature extraction to training to evaluation. While we could not re-create the results of the cascade classifier by Zhu et al. (*1*), we offer a way to improve the classifier by replacing AdaBoost with Adaptive Boosting in the training algorithm. We demonstrated that this improves results on our implementation of their classifier.

6. References

1. Zhu, Q.; Yeh, M.-C.; Cheng, K.-T.; Avidan, S. Fast Human Detection Using a Cascade of Histograms of Oriented Gradients. in *Proc IEEE Computer Vision and Pattern Recognition 2006*, 2, 1491–1498.
2. Masnadi-Shirazi, H.; Vasconcelos, N. Asymmetric Boosting. in *Proc 24th International Conference on Machine Learning*, 609–619, 2007.
3. Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. in *Proc Computer Vision and Pattern Recognition 2005*, **1**, 886–893 vol. 1, Jun 2005.
4. Viola, P.; Jones, M. Rapid Object Detection Using a Boosted Cascade of Simple Features. in *Proc Computer Vision and Pattern Recognition 2001*, vol. 1, I–511-518, 2001.
5. Freund, Y.; Schapire, R. A Decision-Theoretic Generalization of On-Line Learning and An Application to Boosting. in *Computational learning theory*, 23–37, Springer, 1995.
6. Friedman, J.; Hastie, T.; Tibshirani, R. Additive Logistic Regression: A Statistical View of Boosting (With Discussion and a Rejoinder by the Authors). *The Annals of Statistics* **2000**, *28* (2), 337–407.
7. Doll´ar, P. Piotr's Image and Video Matlab Toolbox (PMT). http://vision.ucsd.edu/pdollar/toolbox/doc/index.html (accessed October 2012).
8. Lowe, D. G. Object Recognition from Local Scale-Invariant Features. in *Proc Seventh IEEE International Conference on Computer Vision 1999*, vol. 2, 1150–1157, 1999.

Appendix. Core Source Code

This appendix contains the core source code for the implementation of the algorithm described in this report. The following shows the implementation of the methods for classes representing weak, strong, and cascade classifier classes. This includes the training algorithms.

```
/*
 * Classifiers.cpp
 *
 *  Created on: Apr 3, 2012
 *      Author: creale
 */

#include "Classifiers.h"
#include "IntegralHOGImgFixed.h"
#include "IntegralHOGImg.h"
#include <set>

using namespace std;

void WeakClassifier::train(Mat &trainData, Mat &responses, Mat &weights){
	double cvals[4] = {8,32,128,512};
	params.svm_type = CvSVM::C_SVC;
	params.kernel_type = CvSVM::LINEAR;
	params.C = SVM_C;
	CvMat weightsmat = weights;
	params.class_weights = &weightsmat;
	Mat hyperplaneMax;
	double shiftMax, max_acc = -1000;
	int npos = 0;
	while(responses.at<float>(++npos) == 1);

	double T_plus = 0;
	for(int i = 0; i < npos; i++){
		T_plus+=weights.at<double>(i);
	}
	double T_minus = 0;
	for(int i = npos; i < weights.rows; i++){
		T_minus+=weights.at<double>(i);
	}

	for(int i = 0; i < 4; i++){
		params.C = cvals[i];
		svm.train(trainData, responses, Mat(), Mat(), params);

		Mat pred = Mat::eye(trainData.cols,trainData.cols,CV_32F);
		hyperplane = Mat::zeros(pred.rows,1,CV_32F);
		shift = svm.predict(Mat::zeros(1,pred.rows, CV_32F), true);
		for(int r = 0; r < pred.rows; r++){
			float temp = svm.predict(pred.row(r), true)-shift;
			hyperplane.at<float>(r,0) = temp;
		}

		Mat temp(1, pred.cols, CV_32F);
		for(int c = 0; c < pred.cols; c++)
```

```
                temp.at<float>(c) = rand();
            while(this->predict(temp,true) == 0)
                temp.at<float>(0) = rand();

            if(this->predict(temp, false) != svm.predict(temp, false)){
                hyperplane = hyperplane*-1;
                shift = -shift;
            }

            double bee = 0;
            for(int j = 0; j < npos; j++){
                if(this->predict(trainData.row(j), false) == -1){
                    bee+=weights.at<double>(j);
                }
            }
            double dee = 0;
            for(int j = npos; j < weights.rows; j++){
                if(this->predict(trainData.row(j), false) == 1){
                    dee+=weights.at<double>(j);
                }
            }

            //double weig = getWeight(.5, C1, C2, bee, dee, T_minus, T_plus, .001);
            double acc = 1 - dee - bee;//-getLoss(weig, C1, C2, bee, dee, T_minus, T_plus);

            if(acc > max_acc){
                hyperplaneMax = hyperplane;
                shiftMax = shift;
            }
        }

        hyperplane = hyperplaneMax;
        shift = shiftMax;

        double right = 0, wrong = 0;
        double right1 = 0, wrong1 = 0;
        for(int r = 0; r < trainData.rows; r++){
            bool retDFVal = false;
            float temp = predict(trainData.row(r), retDFVal);
            if(temp == 1)
                if(responses.at<float>(r) == 1)
                    right++;
                else
                    wrong++;
            else
                if(responses.at<float>(r) == 1)
                    wrong1++;
                else
                    right1++;

        }

#ifdef LIBSVM
        svm_free_and_destroy_model(&model);
        delete xspace;
        delete prob.x;
        delete prob.y;
#else
        svm.clear();
#endif
```

```
	accuracy = (right+right1)/(right+right1+wrong+wrong1);
	weight = .5*log(accuracy/(1-accuracy + numeric_limits<double>::epsilon()) + 
numeric_limits<double>::epsilon());

	float detect = right/(right + wrong1);
	float falAlarm = wrong/(right1 + wrong);

	//std::cout << "\tCval: " << maxc << " Acc: " << maxacc << std::endl;
	//std::cout << "Accuracy: " << accuracy << " Det Rate: " << detect << " FA Rate: " << 
falAlarm << std::endl;
}

float WeakClassifier::predict(Mat sample, bool soft){
	float retval = shift;
	for(int i = 0; i < hyperplane.rows; i++)
		retval+=hyperplane.at<float>(i)*sample.at<float>(i);
	if(soft)
		return retval;
	else
		return (retval <= 0) ? 1 : -1;
}

Mat WeakClassifier::predictN(Mat &samples, bool soft){
	Mat retval(samples.rows,1,CV_32F);
	for(int r = 0; r < samples.rows; r++)
		retval.at<float>(r) = predict(samples.row(r), soft);
	return retval;
}

void WeakClassifier::setParams(CvSVMParams params){
	this->params = params;
}

void WeakClassifier::clear(){
	svm.clear();
}

WeakClassifier * WeakClassifier::clone(){
	WeakClassifier *retval = new WeakClassifier;
	retval->accuracy = this->accuracy;
	retval->feat = this->feat;
	retval->hyperplane = this->hyperplane.clone();
	retval->shift = this->shift;
	retval->weight = this->weight;
	return retval;
}

float getWeight(float init_alpha, float c1, float c2, float bee, float dee, float Tminus, 
float Tplus, float epsilon){
	double alpha = init_alpha;
	double func_eval = c1*(bee-Tplus)*exp(-c1*alpha)+c2*(dee-Tminus)*exp(-
c2*alpha)+bee*c1*exp(c1*alpha)+dee*c2*exp(c2*alpha);
	while(abs(func_eval) > epsilon){
		alpha = alpha - func_eval/(-c1*c1*(bee-Tplus)*exp(-c1*alpha)-c2*c2*(dee-
Tminus)*exp(-c2*alpha)+bee*c1*c1*exp(c1*alpha)+dee*c2*c2*exp(c2*alpha));
		func_eval = c1*(bee-Tplus)*exp(-c1*alpha)+c2*(dee-Tminus)*exp(-
c2*alpha)+bee*c1*exp(c1*alpha)+dee*c2*exp(c2*alpha);
	}
	return float(alpha);
}
```

```
float getLoss(float alpha, float c1, float c2, float bee, float dee, float Tminus, float
Tplus){
	return -(bee-Tplus)*exp(-c1*alpha)-(dee-Tminus)*exp(-
c2*alpha)+bee*exp(c1*alpha)+dee*exp(c2*alpha);
}

float StrongClassifier::train(vector<IntegralHOGImgFixed*> &pos_imgs,
vector<IntegralHOGImgFixed*> &neg_imgs, BoxGen &bg, int count){
	vector<FeatBox> fbs = bg.getnRandFeat(NRANDFEATS); //get 250 random features
	vector<WeakClassifier*> weak250; //one classifier for each random feature

	vector<Mat> testSoftResults;
	vector<Mat> testSoftResultsUsed;

	//ground truth of images (1 = human, -1 = non-human)
	Mat classes((int)(pos_imgs.size()+neg_imgs.size()),1, CV_32F, Scalar(-1));
	classes.rowRange(Range(0,pos_imgs.size()-1)) *= -1;

	int ntrainbits = pos_imgs.size() + neg_imgs.size();

	std::cout << "Num training bits: " << ntrainbits << " " << std::endl;

	int npostest = pos_imgs.size();

	Mat weights(ntrainbits,1,CV_64F, Scalar(0.001));
	weights.rowRange(0,pos_imgs.size())*=(neg_imgs.size());
	weights.rowRange(pos_imgs.size(), ntrainbits)*=(pos_imgs.size());
	double sum = 0;
	for(int i = 0; i < weights.rows; i++){
		sum += weights.at<double>(i);
	}
	for(int i = 0; i < weights.rows; i++){
		weights.at<double>(i) /= sum;
	}

	threshold = 0;
	double fi = 1;

	while(fi > FI_MAX){
		fbs.clear();
		fbs = bg.getnRandFeat(NRANDFEATS);
		//testSoftResults.clear();

		//resample distribution based on weights
		//vector<int> trainInds;
		//double unif = 1/((double)classes.rows*SVM_UP);
		//for(int j = 0; j < weights.rows; j++){
		//	for(double k = .5; k*unif < weights.at<double>(j);k++)
		//		trainInds.push_back(j);
		//}

		//create list of classifiers
		for(int i = 0; i < NRANDFEATS; i++){
			WeakClassifier *w = new WeakClassifier();
			Mat tempfeats = extractFeats(fbs[i], pos_imgs, neg_imgs, 2, 2);

//			Mat featsSamp(trainInds.size(), tempfeats.cols, CV_32F);
//			Mat classSamp(trainInds.size(), classes.cols, CV_32F);

//			for(int j = 0; j < trainInds.size(); j++){
//				classSamp.at<float>(j) = classes.at<float>(trainInds[j]);
```

```
//                              //featsSamp.push_back(tempfeats.row(trainInds[j]));
//                              Mat tmp = featsSamp.row(j);
//                              tempfeats.row(trainInds[j]).copyTo(tmp);
//                      }

                        //w->train(featsSamp, classSamp, weights);
                        w->train(tempfeats, classes, weights);
                        w->setFeat(fbs[i]);
                        w->ind = weak250.size();
                        weak250.push_back(w);

                        Mat temp2(ntrainbits, 1, CV_32F);
                        for(int r = 0; r < tempfeats.rows; r++)
                                temp2.at<float>(r) = w->predict(tempfeats.row(r),false);
                        testSoftResults.push_back(temp2);

                        if(i % 25 == 24){
                                std::cout << i+1 << " ";
                                std::cout.flush();
                        }
                }

                double T_plus = 0;
                for(int i = 0; i < pos_imgs.size(); i++){
                        T_plus+=weights.at<double>(i);
                }
                double T_minus = 0;
                for(int i = pos_imgs.size(); i < weights.rows; i++){
                        T_minus+=weights.at<double>(i);
                }

                //update accuracies based on weights
                for(int i = 0; i < weak250.size(); i++){

//                      double acc = 0;
//                      for(int j = 0; j < weights.rows; j++){
//                              if((((testSoftResults[i].at<float>(j) >= 0) ? 1 : -1)
*classes.at<float>(j)) == 1)
//                                      acc += weights.at<double>(j);
//                      }
//                      if(acc >= .5){
//                              weak250[i]->accuracy =
std::max<double>(std::min<double>(acc,.999),.001);
//                              weak250[i]->weight = .5*log(weak250[i]->accuracy/(1-weak250[i]-
>accuracy));
//                      }else{
//                              acc = 1-acc;
//                              weak250[i]->shift = -weak250[i]->shift;
//                              weak250[i]->hyperplane = -weak250[i]->hyperplane;
//                              weak250[i]->accuracy =
std::max<double>(std::min<double>(acc,.999),.001);
//                              weak250[i]->weight = .5*log(weak250[i]->accuracy/(1-weak250[i]-
>accuracy));
//                              testSoftResults[i] = -testSoftResults[i];
//                      }

                        double bee = 0;
                        for(int j = 0; j < pos_imgs.size(); j++){
                                if(testSoftResults[i].at<float>(j) == -1){
                                        bee+=weights.at<double>(j);
                                }
                        }
                        double dee = 0;
                        for(int j = pos_imgs.size(); j < weights.rows; j++){
                                if(testSoftResults[i].at<float>(j) == 1){
                                        dee+=weights.at<double>(j);
                                }
                        }

                        weak250[i]->weight = getWeight(.5, C1, C2, bee, dee, T_minus, T_plus,
.001);
                        //std::cout << T_minus << " " << T_plus << " " << dee << " " << bee << "
" << weak250[i]->weight << std::endl;
                        weak250[i]->accuracy = -getLoss(weak250[i]->weight, C1, C2, bee, dee,
T_minus, T_plus);
                }

                //find best feature available to use
                int ind = 0;
                /*
                double min_fi = 1.01;
                double accur = 0;
                for(int i = 0; i < weak250.size(); i++){
                        //Calculate performance after addition of new weak classifier
                        Mat softCtemp(testSoftResults[0].rows,1,CV_32F,Scalar(0));
                        for(int j = 0; j < testSoftResultsUsed.size(); j++){
                                softCtemp = softCtemp + testSoftResultsUsed[j]*weakCs[j]->weight;
                        }
                        softCtemp = softCtemp + testSoftResults[i]*weak250[i]->weight;

                        //Find threshold to maintain DI_MIN
                        Mat softPosTemp = softCtemp.rowRange(0,npostest);
                        sort(softPosTemp.begin<float>(), softPosTemp.end<float>());
                        float threshTemp = softPosTemp.at<float>(floor(npostest*(1.0 - DI_MIN)))
- .0001;

                        //Calculate fi
                        Mat softNegTemp = softCtemp.rowRange(npostest, softCtemp.rows);

                        float falsepost = 0;
                        for(int r = 0; r < softNegTemp.rows; r++){
                                if(softNegTemp.at<float>(r) >= threshTemp)
                                        falsepost++;
                        }
                        double fi_temp =(falsepost/((double)softNegTemp.rows));
                        if(fi_temp < min_fi){
                                ind = i;
                                min_fi = fi_temp;
                                accur = weak250[i]->accuracy;
                        }else
                                if(fi_temp == min_fi && accur < weak250[i]->accuracy){
                                        ind = i;
                                        accur = weak250[i]->accuracy;
                                }
                }
                */

                double max_acc = -100000;
                for(int i = 0; i < weak250.size(); i++){
                        //std::cout << weak250[i]->accuracy << " ";
                        //if(i % 25 == 24)
                        //std::cout << std::endl;
```

```
			if(max_acc < weak250[i]->accuracy){
				ind = i;
				max_acc = weak250[i]->accuracy;
			}
		}

		WeakClassifier *curr = weak250[ind]->clone();
		testSoftResultsUsed.push_back(testSoftResults[ind]);
		weakCs.push_back(curr);

		//Calculate performance after addition of new weak classifier
		Mat softCombined(testSoftResults[0].rows,1,CV_32F,Scalar(0));
		for(int i = 0; i < testSoftResultsUsed.size(); i++){
			softCombined = softCombined + testSoftResultsUsed[i]*weakCs[i]->weight;
		}

		//Find threshold to maintain DI_MIN
		Mat softPos = softCombined.rowRange(0,npostest).clone();
		sort(softPos.begin<float>(), softPos.end<float>());
		threshold = softPos.at<float>(floor(npostest*(1.0 - DI_MIN))) - .0001;

		//Calculate fi
		Mat softNeg = softCombined.rowRange(npostest, softCombined.rows);

		float falsepos = 0;
		for(int r = 0; r < softNeg.rows; r++){
			if(softNeg.at<float>(r) >= threshold)
				falsepos++;
		}
		fi=(falsepos/((double)softNeg.rows));

		float det = 0;
		for(int r = 0; r < softPos.rows; r++){
			if(softPos.at<float>(r) >= threshold)
				det++;
		}
		det=(det/((double)softPos.rows));

		std::cout << " fi: " << fi << " det: " << det << " thrsh: " << threshold << " 
New acc/weight: " << weak250[ind]->accuracy << " " << weak250[ind]->weight << " " << ind << " 
";
		std::cout << "Feature (x y w h):(" << weak250[ind]->feat.box.x << " " << 
weak250[ind]->feat.box.y << " " << weak250[ind]->feat.box.width << " " << weak250[ind]-
>feat.box.height << ") " << std::endl;

		//update distribution
		double sum = 0;
		for(int i = 0; i < weights.rows; i++){
			double classifierWeight = weak250[ind]->weight;
			//weights.at<double>(i) = weights.at<double>(i)*exp(-
WEIGHT_WEIGHT*classifierWeight*((testSoftResults[ind].at<float>(i) >= 0) ? 1 : -1) * 
classes.at<float>(i));
			if(i < pos_imgs.size())
				weights.at<double>(i) = weights.at<double>(i)*exp(-
C1*classifierWeight*testSoftResults[ind].at<float>(i));
			else
				weights.at<double>(i) = 
weights.at<double>(i)*exp(C2*classifierWeight*testSoftResults[ind].at<float>(i));
			sum += weights.at<double>(i);
		}
		for(int i = 0; i < weights.rows; i++){
			double temp = weights.at<double>(i) / sum;
			weights.at<double>(i) = temp;
		}

		//cleanup
		//vector<int> usedFeatIDS;
		//usedFeatIDS.push_back(weak250[ind]->feat.id);
		//bg.markUsed(usedFeatIDS);

		//for(int i = 0; i < weak250.size(); i++)
		//if(i != ind)
		//delete weak250[i];
		//weak250.clear();
	}

	std::stringstream filename;
	filename << "ccout\\train_" << count << ".txt";
	ofstream outFile(filename.str());
	for(int i = 0; i < testSoftResults.size(); i++){
		for(int j = 0; j < testSoftResults[i].rows; j++){
			outFile << testSoftResults[i].at<float>(j);
			if(j < testSoftResults[i].rows-1)
				outFile << " ";
			else if (i < testSoftResults.size()-1)
				outFile << std::endl;
		}
	}
	outFile.close();

	vector<IntegralHOGImgFixed*> posList = loadPos(false);
	vector<IntegralHOGImgFixed*> negList = loadNeg(false, 5, NULL);
	std::stringstream filename2;
	filename2 << "ccout\\test_" << count << ".txt";
	ofstream outFile2(filename2.str());
	for(int i = 0; i < weak250.size(); i++){
		for(int j = 0; j < posList.size(); j++){
			float * tempFeat = posList[j]->extractFeature(weak250[i]->feat.box,2,2);
			Mat sample(1, 36, CV_32F, tempFeat);
			outFile2 << weak250[i]->predict(sample,false) << " ";
			delete tempFeat;
			if(i == weak250.size()-1)
				delete posList[j];
		}
		for(int j = 0; j < negList.size(); j++){
			float * tempFeat = negList[j]->extractFeature(weak250[i]->feat.box,2,2);
			Mat sample(1, 36, CV_32F, tempFeat);
			if(j < negList.size()-1)
				outFile2 << weak250[i]->predict(sample,false) << " ";
			else if(i <  weak250.size()-1)
				outFile2 << weak250[i]->predict(sample,false) << std::endl;
			else
				outFile2 << weak250[i]->predict(sample,false);
			delete tempFeat;
			if(i == weak250.size()-1)
				delete negList[j];
		}
	}
	outFile2.close();
	posList.clear();
	negList.clear();
	for(int i = 0; i < weak250.size(); i++){
		delete weak250[i];
	}
	weak250.clear();

	return (float)fi;
}

bool StrongClassifier::predict(IntegralHOGImgFixed &ihif){
	double result = 0;
	for(WeakClassifier * wc: weakCs){
		FeatBox fb = wc->feat;
		float * feat = ihif.extractFeature(fb.box,2,2);
		Mat sample(1, 36, CV_32F, feat);
		result += wc->predict(sample, false)*wc->weight;
		delete feat;
	}

	return (result >= threshold);
}

float StrongClassifier::predict(IntegralHOGImg &ihi, Point p){
	double result = 0;
	for(WeakClassifier * wc: weakCs){
		FeatBox fb = wc->feat;
		float * feat = ihi.extractFeature(Rect(p.x+fb.box.x, p.y+fb.box.y, fb.box.width, 
fb.box.height),2,2);
		Mat sample(1, 36, CV_32F, feat);
		result += wc->predict(sample, false)*wc->weight;
		delete feat;
	}

	return result - threshold;
}

Mat StrongClassifier::extractFeats(FeatBox fb, vector<IntegralHOGImgFixed*> pos_imgs, 
vector<IntegralHOGImgFixed*> neg_imgs, int xcells, int ycells){
	int featlen = NBINS*xcells*ycells;
	int nfeats = pos_imgs.size() + neg_imgs.size();
	Mat retVal(nfeats, featlen, CV_32F);

	for(int r = 0; r < pos_imgs.size(); r++){
		float *temp = pos_imgs[r]->extractFeature(fb.box,xcells,ycells);
		for(int j = 0; j < featlen; j++){
			retVal.at<float>(r,j) = temp[j];
		}
		delete temp;
	}

	for(int r = 0; r < neg_imgs.size(); r++){
		float *temp = neg_imgs[r]->extractFeature(fb.box,xcells,ycells);
		for(int j = 0; j < featlen; j++){
			retVal.at<float>(r+pos_imgs.size(),j) = temp[j];
		}
		delete temp;
	}

	return retVal;
}
```

```
vector<IntegralHOGImgFixed*> loadPos(bool train){
        vector<IntegralHOGImgFixed*> pos_imgs;
        string line;
        string listfile;

        if(train)
                listfile = "train\\pos.lst";
        else
                listfile = "test\\pos.lst";
        ifstream myfile (listfile);
        if (myfile.is_open()){
                while (myfile.good()){
                        getline (myfile,line);
                        Mat img = imread(line,0);
                        IntegralHOGImgFixed * hogimg = new IntegralHOGImgFixed(img);
                        pos_imgs.push_back(hogimg);
                }
                myfile.close();
        }else cout << "Unable to open file 1";

        return pos_imgs;
}

vector<IntegralHOGImgFixed*> loadNeg(bool train, int nperi, CascClassifier *cc){
        vector<IntegralHOGImgFixed*> neg_imgs;
        string line;
        string listfile;

        if(train)
                listfile = "train\\neg.lst";
        else
                listfile = "test\\neg.lst";

        int counter = 0;
        ifstream myfile2 (listfile);
        if (myfile2.is_open()){
                while (myfile2.good()){
                        getline (myfile2,line);
                        Mat img = imread(line,0);
                        if(cc == NULL){
                                if(img.rows >= 130 && img.cols >= 66){
                                        for(int i = 0; i < nperi; i++){
                                                int x = rand() % (img.cols-66);
                                                int y = rand() % (img.rows-130);
                                                Mat subimg = img(Range(y,y+130), Range(x,x+66));
                                                IntegralHOGImgFixed *hogimg = new
IntegralHOGImgFixed(subimg);
                                                neg_imgs.push_back(hogimg);
                                        }
                                }
                        }else{
                                int count = 0;
                                Mat *imgptr = &img;
                                int newr = img.rows;
                                int newc = img.cols;
                                while(count < nperi && imgptr->rows > 130 && imgptr->cols > 66){
                                        vector<Rect> fps = cc->findFalsePos(*imgptr);
                                        for(Rect rect : fps){
                                                int x = rect.x-1; //we want a one pixel cushion to
create ihif
                                                int y = rect.y-1;
                        if(count < nperi){
                            Mat subimg = imgptr-
>rowRange(y,y+130).colRange(x,x+66);
                            IntegralHOGImgFixed *hogimg = new
IntegralHOGImgFixed(subimg);
                            neg_imgs.push_back(hogimg);
                            count++;
                        }
                    }

                    newr = newr/2;
                    newc = newc/2;
                    Mat *downImg = new Mat(newr, newc, imgptr->type());
                    resize(*imgptr, *downImg, downImg->size(), 0, 0,
CV_INTER_LINEAR);
                    if(imgptr->rows != img.rows)
                        delete imgptr;
                    imgptr = downImg;
                }
                delete imgptr;
            }
        }
        myfile2.close();
    }else cout << "Unable to open file 2";
    return neg_imgs;
}

void CascClassifier::train(){
    time_t rawtime;
    time(&rawtime);
    tm *timeinfo = localtime(&rawtime);
    int month = timeinfo->tm_mon + 1;
    int day = timeinfo->tm_mday;
    int year = timeinfo->tm_year+1900;

    vector<IntegralHOGImgFixed*> pos_imgs = loadPos(true);
    vector<IntegralHOGImgFixed*> neg_imgs;

    std::cout << "Positive Images Loaded" << std::endl;

    //create feature generator
    int widths[] = {12,16,20,24,28,32,36,40,48,56,64};
    vector<int> wid;
    for(int w = 0; w < 11; w++)
        wid.push_back(widths[w]);
    vector<int> ratios;
    ratios.push_back(1);
    ratios.push_back(0);
    ratios.push_back(-1);
    BoxGen bg(64, 128, wid, ratios);

    double fi = 1;
    double di = 1;
    int count = 0;
    int nperi = 5;

    std::stringstream pfilename;
    pfilename << "ccdata\\" << month << "_" << day << "_" << year << "_param" << ".txt";
    saveParamsToFile(pfilename.str());

    while(fi > F_TARG || count < 40){
```

```
                neg_imgs = loadNeg(true, nperi, this);
                if(neg_imgs.size() + pos_imgs.size() < 7000)
                        nperi*=2;

                std::cout << "Negative Images Loaded" << std::endl;

                vector<IntegralHOGImgFixed*> neg_imgs2;

                //take subset of negative samples
                for(int i = 0; i < neg_imgs.size(); i++){
                        if((rand() % neg_imgs.size()) < 2*pos_imgs.size())
                                neg_imgs2.push_back(neg_imgs[i]);
                }

                StrongClassifier * sc = new StrongClassifier();
                float fi_ = sc->train(pos_imgs, neg_imgs2, bg, count);
                strongCs.push_back(sc);
                fi*=fi_;
                di*=DI_MIN;

                int det = 0;
                for(int i = 0; i < pos_imgs.size(); i++){
                        if(sc->predict(*pos_imgs[i]))
                                det++;
                }

                std::cout << "Trained Strong Classifier " << strongCs.size() << " Number of weak
classifiers: " << sc->weakCs.size() << " fi: " << fi << std::endl;
                std::cout << "\tDet: " << det << std::endl;

                std::stringstream filename;
                filename << "ccdata\\" << month << "_" << day << "_" << year << "_n" << count <<
".txt";
                saveToFile(filename.str());

                count++;

                for(IntegralHOGImgFixed * ihif : neg_imgs)
                        delete ihif;
                neg_imgs.clear();
        }

        for(IntegralHOGImgFixed * ihif : pos_imgs)
                delete ihif;
}

bool CascClassifier::predict(IntegralHOGImgFixed &ihif){
        for(int i = 0; i < strongCs.size(); i++)
                if(!strongCs[i]->predict(ihif))
                        return false;
        return true;
}

float CascClassifier::predict(IntegralHOGImg &ihi, Point p){
        float sum = 0.0001; //default positive
        for(int i = 0; i < strongCs.size(); i++){
                float pred = strongCs[i]->predict(ihi, p);
                if(pred < 0)
                        return -1;
                else
                        sum+=pred;
```

```
	}
	return sum;
}

bool tieBreak(int r1, int c1, int r2, int c2){
	return (r1 * 1000 + c1) < (r2 * 1000 + c2);
}

int rectDom(RectConfidence rc1, RectConfidence rc2){
	Point tl1 = Point(rc1.rect.x, rc1.rect.y);
	Point br1 = Point(rc1.rect.x+rc1.rect.width, rc1.rect.y+rc1.rect.height);
	Point tl2 = Point(rc2.rect.x, rc2.rect.y);
	Point br2 = Point(rc2.rect.x+rc2.rect.width, rc2.rect.y+rc2.rect.height);

	Rect overlap;
	overlap.x = max(tl1.x, tl2.x);
	overlap.y = max(tl1.y, tl2.y);
	overlap.width = min(br1.x, br2.x) - overlap.x;
	overlap.height = min(br1.y, br2.y) - overlap.y;
	if(min(overlap.height, overlap.width) <= 0 ) //not overlapping at all
		return 0;
	else

	if(((double)overlap.width*overlap.height)/min(rc1.rect.width*rc1.rect.height,rc2.rect.w
idth*rc2.rect.height) >= .65)
			return (rc1.conf >= rc2.conf) ? 1 : 2;
		else
			return 0;
}

void CascClassifier::findPeople(Mat &img, vector<RectConfidence> &alldets,
vector<RectConfidence> &retval){
	double scale = 1.0;
	float width = img.cols;
	//namedWindow("TestWindow2", CV_WINDOW_AUTOSIZE);
	int w, h;

	while((w = (int)(img.cols*scale)) >= 66 && (h = (int)(img.rows*scale)) >= 130){
		double scalex = ((double)w)/img.cols;
		double scaley = ((double)h)/img.rows;

		Mat scaledImg(h,w,img.type());

		resize(img, scaledImg, scaledImg.size(), 0, 0, CV_INTER_LINEAR);

		IntegralHOGImg ihi(scaledImg,9);
		for(int r = 0; r < ihi.rows-130; r+=8){
			for(int c = 0; c < ihi.cols-66; c+=8){
				//x = c, y = r
				float conf = predict(ihi,Point(c,r));
				if(conf != -1){
					Rect re((int)c/scalex, (int)r/scaley, (int)64/scalex,
(int)128/scaley);
					RectConfidence rc;
					rc.rect = re;
					rc.conf = conf;
					alldets.push_back(rc);
				}
			}
		}
		scale/=1.05;
	}

	//Non Maximum Suppression
	list<int> inds;
	for(int i = 0; i < alldets.size(); i++)
		inds.push_back(i);

	list<int>::iterator it1 = inds.begin();
	while(it1 != inds.end()){
		list<int>::iterator it2 = it1;
		it2++;

		while(it2 != inds.end()){
			int dom = rectDom(alldets[*it1], alldets[*it2]);
			if(dom == 0)
				it2++;
			else if(dom == 1)
				inds.erase(it2++);
			else
				break;
		}

		if(it2 == inds.end())
			it1++;
		else
			inds.erase(it1++);
	}

	for(int ind : inds)
		retval.push_back(alldets[ind]);
}

int primes[50] = {3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833,
3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907,
			  3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001,
4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057,
			  4073, 4079, 4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139};

vector<Rect> CascClassifier::findFalsePos(Mat &img){

	vector<Rect> retval;
	IntegralHOGImg *hogimg = new IntegralHOGImg(img,9);

	int rprime = primes[rand()%50];
	int cprime = primes[rand()%50];

	for(int r = 1; r < img.rows-130 && retval.size() < 10; r++){
		for(int c = 1; c < img.cols-66 && retval.size() < 10; c++){
			int r1 = (((r+c)*rprime) % (img.rows-131)) + 1;
			int c1 = (((c)*cprime) % (img.cols-67)) + 1;
			if(this->predict(*hogimg, Point(c1,r1)) > 0)
				retval.push_back(Rect(c1,r1,64,128));
		}
	}

	delete hogimg;

	return retval;
}
```

```
void CascClassifier::saveToFile(String filename){
	ofstream outFile(filename);

	if(outFile.is_open()){
		outFile << strongCs.size() << std::endl;
		for(StrongClassifier * sc : strongCs){
			outFile << sc->threshold << std::endl;
			outFile << sc->weakCs.size() << std::endl;
			for(WeakClassifier * wc : sc->weakCs){
				outFile << wc->shift << std::endl;
				outFile << wc->weight << std::endl;
				outFile << wc->feat.box.x << std::endl;
				outFile << wc->feat.box.y << std::endl;
				outFile << wc->feat.box.width << std::endl;
				outFile << wc->feat.box.height << std::endl;
				for(int i = 0; i < wc->hyperplane.rows; i++){
					outFile << wc->hyperplane.at<float>(i) << std::endl;
				}
			}
		}
	}

	outFile.close();
}

void CascClassifier::saveParamsToFile(String filename){
	ofstream outFile(filename);

	if(outFile.is_open()){
		outFile << "FI_MAX: " << FI_MAX << std::endl;
		outFile << "F_TARG: " << F_TARG << std::endl;
		outFile << "DI_MIN: " << DI_MIN << std::endl;
		outFile << "NRANDFEATS: " << NRANDFEATS << std::endl;
		outFile << "SVM_C: " << SVM_C << std::endl;
		outFile << "C1: " << C1 << std::endl;
		outFile << "C2: " << C2 << std::endl;
		outFile << "OVERLAP: " << OVERLAP << std::endl;
	}

	outFile.close();
}

void CascClassifier::readFromFile(String filename){
	ifstream inFile(filename);

	String line;
	int nwc, nsc;
	if(inFile.is_open()){
		getline(inFile, line);
		istringstream(line) >> nsc;
		for(int sci = 0; sci < nsc; sci++){
			strongCs.push_back(new StrongClassifier());
			getline(inFile, line);
			istringstream(line) >> strongCs[sci]->threshold;
			getline(inFile, line);
			istringstream(line) >> nwc;
			for(int wci = 0; wci < nwc; wci++){
				strongCs[sci]->weakCs.push_back(new WeakClassifier());
				getline(inFile, line);
				istringstream(line) >> this->strongCs[sci]->weakCs[wci]->shift;
				getline(inFile, line);
```

```
                                istringstream(line) >> this->strongCs[sci]->weakCs[wci]->weight;
                                getline(inFile, line);
                                istringstream(line) >> this->strongCs[sci]->weakCs[wci]-
>feat.box.x;

                                getline(inFile, line);
                                istringstream(line) >> this->strongCs[sci]->weakCs[wci]-
>feat.box.y;

                                getline(inFile, line);
                                istringstream(line) >> this->strongCs[sci]->weakCs[wci]-
>feat.box.width;

                                getline(inFile, line);
                                istringstream(line) >> this->strongCs[sci]->weakCs[wci]-
>feat.box.height;

                                strongCs[sci]->weakCs[wci]->hyperplane = Mat::zeros(36,1,CV_32F);
                                for(int r = 0; r < 36; r++){
                                        getline(inFile, line);
                                        istringstream(line) >> strongCs[sci]->weakCs[wci]-
>hyperplane.at<float>(r);

                                }
                        }
                }
        }

        inFile.close();
}
```

NO. OF COPIES	ORGANIZATION
1 (PDF)	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
4 (PDFS)	US ARMY CERDEC NVESD ATTN L GRACEFFO ATTN M GROENERT ATTN J HILGER ATTN J WRIGHT
2 (PDFS)	US ARMY AMRDEC ATTN RDMR WDG I J MILLS ATTN RDMR WDG S D WAAGEN
2 (PDFS)	US ARMY RSRCH OFFICE ATTN RDRL ROI C L DAI ATTN RDRL ROI M J LAVERY
18 (PDFS)	US ARMY RSRCH LAB ATTN IMAL HRA MAIL & RECORDS MGMT ATTN RDRL CIO LL TECHL LIB ATTN RDRL SE P PERCONTI ATTN RDRL SES J EICKE ATTN RDRL SES M D'ONOFRIO ATTN RDRL SES N NASRABADI ATTN RDRL SES E R RAO ATTN RDRL SES E A CHAN ATTN RDRL SES E H KWON ATTN RDRL SES E S YOUNG ATTN RDRL SES E J DAMMANN ATTN RDRL SES E D ROSARIO ATTN RDRL SES E H BRANDT ATTN RDRL SES E S HU ATTN RDRL SES E M THIELKE ATTN RDRL SES E P RAUSS ATTN RDRL SES E P GURRAM ATTN RDRL SES E C REALE

INTENTIONALLY LEFT BLANK.